CS 6200 Information Retrieval
Seattle/Fall 2019

Assignment 3 Ranked Retrieval

Announced Oct 21, 2019
Due: Nov 4, 2019, 9am Pacific

In Assignment 1, you implemented a simple web crawler, and used it to crawl a portion of Wikipedia. In Assignment 2, you then transformed crawled documents and got them ready to be indexed. You also wrote code to create a simple inverted index system, using which you wrote out an index for these documents and tested the indexing.

In Assignment 3, you will **develop a simple retrieval & ranking system**. In a later assignment, you will create an evaluation component. Slowly but surely, you are working on several components of a simple search engine. Keep in mind that you may be using and building further upon your code and using the output of the code in the next assignment(s). You may also need to slightly modify your code as we get into later assignments. So, make sure to write good, maintainable code.

Do not use any package or library that scores or ranks documents. You must write your own code, and ensure it is well-documented. ***As always, keep in mind the academic integrity rules we discussed.***

Now here are the inputs and outputs, and what's to be done in this assignment.

Inputs and Outputs

1. The input parameters to the ranked retrieval component will be:
    a. **IndexFolderName**, the name of a directory (folder) where you stored the index files that you created in Assignment 2, namely:
        i. A file *TermIDFile* that contains data structure(s) that maps terms into TermIDs and stores document frequencies with the TermIDs.
        ii. A file *DocumentIDFile* that contains data structure(s) that maps DocumentIDs to document names, and stores the document length with each DocumentID.
        iii. A file named *InvertedIndex* that stores a collection of inverted lists, one for each TermID. Each inverted list is a list of postings, where each posting contains a DocumentID and the relevant term frequency.

    b. A folder named **ContentFolderName** where there is a set of documents that you crawled in Assignment 1. If you saved the contents of the 1000 documents you crawled for Assignment 1, you can use those document contents. Otherwise, you may have to run the crawler program again to get the 1000 documents into such a folder and use the contents from that folder. Make sure the folder of documents is consistent with the index files you created in Assignment 2.

    c. A text file **Queries.txt** with at least 10 queries, one query per line.

    d. An integer parameter **K**, the maximum number of results to return for each query.

2. The outputs to be submitted will be as follows:

    a. One or more **program files** which implement this assignment, including the functions (**UseIndex**) to use the index that you wrote as part of Assignment 2.
    b. The **RunRankedRetrieval** script or Python code (see below)
    c. A file named **Output.txt** (or **Output.json**, see below) which is the output of the RunRankedRetrieval script (see Part 1 below for the format of this output file)
    d. A file named **requirements.txt** which contains the list of installed packages, in requirements format. You can get this using the command
    ```
    pip freeze > requirements.txt
    ```
    e. A file named **README.txt** which contains:
        i. A one or two sentence answer each to both the questions: What was the most difficult part of this assignment? What was the easiest part?
        ii. Optional: Any additional details about the format of the 3 index files, if there's something special you have done (e.g. serializing etc.).
        iii. Optional: Any additional information you may wish to provide about running the script in (b) above.

Part 1 Code Requirements

Broadly, in this section, you will create code to do the following: First, read in a file of queries. Then, run each query over the documents in the index you created in Assignment 2, rank all documents in the index for each query using cosine similarity and vector space ranking, and write out the top K results in the prescribed format.

Specifically, you will:

1. Run the same tokenization and data transformation on each query that you ran on documents in Assignment 2. For this assignment (since you did NOT use stop-words or do any stemming for the documents in the index), do not use stop-words or stemming on the queries.
2. Keep track of the transformed query tokens.
3. Represent each transformed query as a normalized, weighted tf-idf vector (**lnc.ltc** weighting)
4. Represent each document similarly as a normalized, weighted tf-idf vector (**lnc.ltc** weighting)
5. For each query,
    a. Compute the cosine similarity score for the query vector and each document vector.
    b. Rank (sort) the documents with respect to the query by the cosine similarity score.
    c. Return the top *K* documents as results, in descending order of the cosine score, as detailed below.

Use the files in the **IndexFolderName** folder to get document frequencies for each term. Get term frequencies for each term and document of interest from the postings lists in the index. Remember to normalize vectors. You will make use of the functions you wrote in Assignment 2, in the file **UseIndex.py**, to implement the ranked retrieval.

For **each** query, write out the following information in **Output.txt**:

1. Line 1: raw query
2. Line 2: tokenized, data transformed, query
3. A set of up to K results, *each of which is in the following format:*
    a. Line r1: DocumentID of the result file <tab> document name. You'll need this in the next assignment on evaluation.
    b. Line r2: A 'snippet' containing the first 200 bytes from the document, retrieved from the data folder **ContentFolderName** containing all the crawled documents. [Optional: If you can go past the html, and get to the first 200 bytes of text, that would be great.]
    c. Line r3: Cosine similarity score between this document and the current query. Make sure the results are ordered in descending order of this score, for each query.
    d. For each word/term in the query, its contribution to the cosine score, in the following format: "term1: contribution1 ; term2: contribution 2 …" For instance, for the example in the "Retrieval Models 1" lecture, we would output:
       ```
       auto: 0.0; best: 0.0; car: 0.27; insurance: 0.53
       ```
    e. Line r4: blank
4. Another two blank lines at the end of up to K results for each query.

If you prefer, you can write out the results as a JSON file, **Output.json,** using an appropriate format.

Now you have a search engine that delivers ranked results!! Congrats!

Part 2. Running the Code

1. When the coding is completed, run the **RunRankedRetrieval** script or Python code which invokes the code described in Part 1, with the following parameters:
    a. **IndexFolderName**, the name of a directory (folder) where you stored the index files.
    b. **ContentFolderName:** the folder where you have the contents of the documents you crawled.
    c. A text file **Queries.txt** with at least 10 queries, one per line. The queries must include a few 1-word queries, a few 2-word queries, and a few 3-or-more-word queries. The queries must be chosen so that you can find at least two and ideally more results for each query, from the documents you indexed in Assignment 2. Do not use queries that consist of nothing but stop words. Make it fun and use interesting queries!
    d. Parameter **K**, set to 5.

Assemble the outputs, as specified in the Inputs & Outputs section above. When you have all your files ready, put them all in a folder, zip up the folder, and name the zip file using the format: **yourFirstName_yourLastName_Assign3** . Submit this zip file via Blackboard.